IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 7, 2021, accepted September 16, 2021, date of publication September 20, 2021,
date of current version September 27, 2020.

Digital Object Identifier 10.1109/ACCESS.2021.3113891

The New TCP Modules on the Block: A Performance Evaluation of TCP Pacing and TCP Small Queues

CARLO AUGUSTO GRAZIA, (Member, IEEE), MARTIN KLAPEZ, AND MAURIZIO CASONI, (Senior Member, IEEE)
Department of Engineering Enzo Ferrari, University of Modena and Reggio Emilia, 41125 Modena, Italy
Corresponding author: Carlo Augusto Grazia (carloaugusto.grazia@unimore.it)

ABSTRACT Google and the Bufferbloat community have designed several solutions to reduce Internet latency in recent years, involving different TCP-IP stack layers. One of these solutions is named TCP Small Queues (TSQ) and reduces a TCP flow latency by controlling the number of packets that each TCP socket can enqueue in the sender node. It works in conjunction with TCP Pacing (TP), which affects the actual TSQ size as a function of the TCP rate. This paper analyzes TSQ and TP's performance through real-system tests over different networks' bottlenecks, emphasizing Wi-Fi technologies, where their behavior strongly affects the Wi-Fi frame aggregation mechanism.

INDEX TERMS Bufferbloat, latency, TCP small queues.

I. INTRODUCTION

Internet latency issues have been exposed under a magnifying glass in recent years, involving big players like Google and the Bufferbloat community to design novel algorithms for several TCP-IP layers. To name one, the Bufferbloat community has developed the hybrid packet scheduler and queue manager Flow Queue Controlled Delay (FQ-CoDel) [1]. FQ-CoDel has quickly become the standard queueing discipline for many Linux-based end nodes and routers. At the same time, even Google has designed a couple of remarkable algorithms like TCP Bottleneck Bandwidth and Round-trip propagation time (BBR) [2] and TCP Small Queues (TSQ). The former is a transport layer solution, and the latter is a cross-layering solution, considering the TCP-IP stack. Unlike FQ-CoDel, which can be deployed on any node of the path, BBR and TSQ are designed exclusively for the end nodes.

To the best of our knowledge, there is a lack of scientific contributions related to TSQ and TP. In particular, we report a lack in the performance analysis of TSQ in conjunction with these new TCP-IP solutions involving all the stack layers. The TSQ module alone is reported in a few scientific contributions [3]–[6] involving wired networks, but without investigating the main purpose of the mechanism, i.e., to reduce latency without reducing the throughput. For what concerns wireless environments, instead, other few scientific contributions [7]–[9], dealing with LTE and Wi-Fi technologies, report some insight on latency reduction by tuning the TSQ size. Unfortunately, these previous works deal with old Linux kernels, in which TSQ was operating statically by imposing an amount of data to be enqueued, instead of controlling the amount of data to be enqueued dynamically based on the current data delivery rate. Moreover, none of these cited works accommodate a broader analysis of TSQ and TP's interaction in real and up-to-date environments. Indeed, to properly investigate the TSQ impact on latency reduction, a complete system involving TCP BBR and FQ-CoDel must also be considered concerning the bottleneck position in the network path. The sole literature contributions combining TSQ and TP, to the best of our knowledge, are [10] and [11]: in the former, the authors analyzed the CPU impact of these solutions, discussing software bottlenecks more than network bottlenecks, while in the latter, the authors investigated the behavior of different TCP congestion controls regarding an hybrid bottleneck network which involves only Wi-Fi 6 environments.

The contribution of this paper is (i) the description of the TSQ and TP, contextualized in the latest Long-Term Support (LTS) version of the Linux kernel 5.10-lts with the other involved TCP-IP solutions on both wired and wireless

The associate editor coordinating the review of this manuscript and approving it for publication was Barbara Masini.

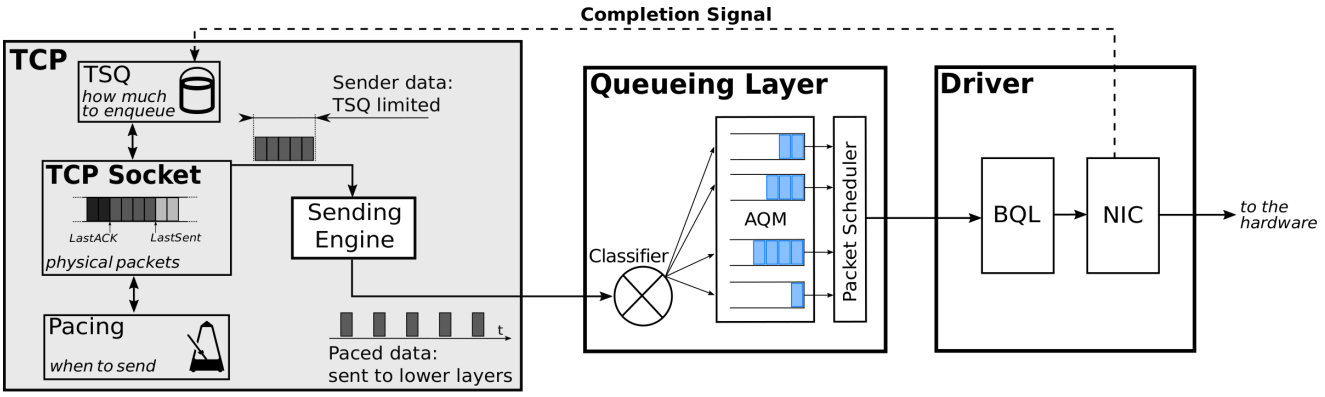


FIGURE 1. TCP-IP linux stack.

scenarios, and (ii) a real-test analysis of the TSQ and TP performance under different network technologies. The paper is structured as follows: Section II describes the latest TCP-IP Linux stack and Section III illustrates the testbed used to collect our data. Finally, Section IV shows the results and Section V concludes the paper.

II. TCP-IP LINUX STACK

The current and up-to-date TCP-IP Linux stack is depicted in Figure 1. We report the three main blocks involved in a TCP flow transmission with the whole TCP transport block on the left, the Queueing Layer corresponding to the TCP-IP networking layer in the middle and the host-to-network Driver block on the right. The TCP congestion control algorithm role did not change recently, so the TCP socket is still calculating the congestion window (CWND) and dealing with the ACK reception according to the algorithm used. The most significant change in recent years has been in the way packets are delivered by the TCP socket, now regulated by TSQ and TP, also shown in Figure 1. Once the TCP socket delivers the packets, they are enqueued in the lower layers. The Queueing Layer, depicted in the middle of Figure 1, deploys a standard FQ-CoDel algorithm, which is the default solution in recent kernels with many Linux distributions [1]. Once the scheduler delivers the packets, they move to the last block, where the driver firmware implements the last hardware queue before moving to the physical medium channel. Once a packet is physically transmitted, a completion signal is cross-passed to the TSQ algorithm.

Algorithm 1 TCP Pacing Rate

Input: **TCP_SOCKET** sk, **int** base_{RTT};

```
1: int rate = mss * sk→cwnd / base_RTT;
2: if sk→cwnd < sk→ssthresh / 2 then
3:     rate *= tcp_pacing_ss_ratio;        // SlowStart phase
4: else
5:     rate *= tcp_pacing_ca_ratio;        // Cong.Avoid. phase
6: end if
```

A. TP AND TSQ

The most significant change experienced by the TCP-IP stack in recent years has been the introduction of TP and TSQ. The cooperative work of these two TCP submodules strongly impacts the way packets are delivered by the TCP socket, affecting the TCP RTT and the system latency. TP is controlled by two system variables, i.e., `tcp_pacing_ss_ratio` and `tcp_pacing_ca_ratio`, used in the slow start and the congestion avoidance phases, respectively, as reported in Algorithm 1. The TCP socket's final TCP-paced rate to deliver data is then adjusted with a pacing ratio that changes according to the TCP transmission phase. By default, `tcp_pacing_ss_ratio` is equal to 2 in the slow-start phase, and `tcp_pacing_ca_ratio` is equal to 1.2 in the congestion avoidance phase. This means that the TCP flow doubles the slow-start phase rate and increases it by 20% in the congestion-avoidance phase. This mechanism allows probing for more bandwidth without forming excessive bursts of packets in the path's network queues.

Algorithm 2 TCP Small Queue

Input: **TCP_SOCKET** sk;

```
1: int limit;
2: limit = max(2 * sk→pktsize, sk→tcp_pacing_rate ≫ 10);
3: limit = min(limit, tcp_limit_output_bytes);
```

On the other hand, the TCP paced rate is used to calculate, in conjunction with the TSQ mechanism, the number of packets that a TCP socket can enqueue in the sender stack. This quantity is a dynamic value described in Algorithm 2. According to the algorithm, the TSQ limit is always higher than a minimum amount of 2 packets and lower than a maximum amount of `limit_output_bytes` bytes (128 KB by default). The dynamic limit moves through these two bounds and is the amount of data that corresponds to a latency equal to 1 ms by default. Algorithm 2 clarifies this behavior: the dynamic amount of data that can be enqueued is calculated

through $sk \rightarrow$tcp_pacing_rate $\gg 10$, which is a 10-bit shift of the current pacing rate, that corresponds to the amount of data transmitted in 1 ms at the current paced rate. This mechanism helps the sender congestion control mitigate the queueing delay occurring inside the node and accurately calculate RTTs. Concluding, the bit shift quantity changes the latency introduced by TSQ, while the TP ratio changes the TSQ limit size. It is important to notice that BBR operates with a customized TCP pacing mechanism used in its finite state machine. BBR, indeed, is the sole congestion control to ignore the global TP module and does not react to the global `tcp_pacing_ratio` variable changes. BBR deploys a specific finite-state machine, reported in Figure 2, where the pacing rate is computed as a function of the bottleneck bandwidth. BBR estimates the parameters in the finite-state machine, in kernel-space, and cannot be modified by the user.

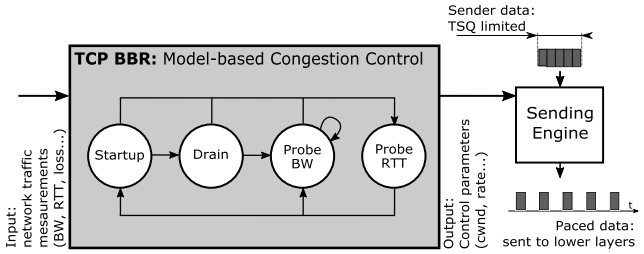


FIGURE 2. Linux TCP BBR block.

B. QUEUEING LAYER AND DRIVER

The default structure of FQ-CoDel, reported in Figure 1, works as follows: a separate software queue serves each TCP flow, and each queue is managed by the CoDel algorithm to control the latency and is served in a round-robin fashion. The default CoDel threshold is set at 5 ms, which means that packets with sojourn time greater than the threshold will be dropped at the dequeue stage. The queueing discipline in novel Linux systems is managed, by default, with the `tc` tool, which allows configuring the characteristics of the networking layer. The Queueing layer and the Network Interface Card (NIC) Driver blocks are strongly coupled in their behavior, and Figure 1 represents a simple scenario in which a single hardware queue is present. The driver also implements the Byte Queue Limit (BQL) for all the hardware queues, which is the last algorithm to control the global latency of the system [12]. The BQL mechanism tries to store enough data to avoid starvation and, simultaneously, to avoid accumulating excessive data increasing the latency. The BQL algorithm is not tested in our paper, and the drivers' default configurations are maintained. A considerable change imposed by the usage of a wireless Atheros NIC equipped with the `ath9k` driver, as is the case of our tests, is that it implements the FQ-CoDel mechanism directly in the firmware, disabling the queueing discipline layer when the driver is used [13]. Thus, it impacts the maximum aggregation size of the NIC due to the 5 ms limit imposed by FQ-CoDel.

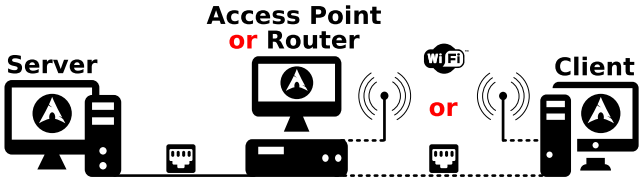


FIGURE 3. Physical testbed.

TABLE 1. Bottlenecks configuration.

Conf. Name	Description
L10M	local C↔R limited at 10 Mbit/s
R10M	remote S↔R limited at 10 Mbit/s
L100M	local C↔R limited at 100 Mbit/s
R100M	remote S↔R limited at 100 Mbit/s
LR1000M	all the connections hardware limited at 1000 Mbit/s

III. TESTBED

To validate the performance of TSQ and TP, we designed the testbed reported in Figure 3. Our testbed is composed of 3 nodes, resembling a classical internet connection with Wi-Fi access for the Client `C` and a wired backhaul for the Server `S`. In the middle between `C` and `S` there is the Router `R` or Access Point `AP`, which interconnects the endpoints and implements a wired testbed or a wireless one, respectively. A Gigabit Ethernet supports the connection between `S` and `R/AP`. In contrast, the connections between `R/AP` and `C` are a Gigabit Ethernet and an IEEE 802.11n, both configured with PCIe devices. These embed the `BCM5761` chipset in the wired testbed and the `AR9580` chipset in the wireless one. The bottleneck segment is software-defined to be local (the link between `C` and `R`) or remote (the link between `S` and `R`) in the wired testbed. The difference between local and remote bottlenecks resembles different possible laboratories and home connections, allowing us to focus on widely different possible real networks by only controlling the few testbed nodes. To deploy a configurable wired bottleneck, we used the `tc` Linux package and implemented a hierarchical token bucket (HTB) queueing discipline, coupled with the default FQ-CoDel on the Router interfaces. It is important to notice that `tc` allows for traffic shaping through HTB, i.e., imposing a specific delivery rate for an interface, without compromising the scheduling and AQM algorithms behaviors, running in cascade to HTB. This is not true if the bottleneck is modeled also imposing specific delay or packet losses thorough the `netem` queueing discipline, which must be used in mutual exclusion to the other queueing disciplines such as FQ-CoDel. Anyway, this is not the case in our discussion since we focus only on the bottleneck bandwidth characteristic. We defined 5 possible different bottleneck configurations, summarized in Table 1. Instead, the bottleneck is the physical wireless interface in the wireless testbed, reflecting a standard Wi-Fi home access network with a Gigabit Ethernet backhaul. All the nodes run an Arch Linux distribution with a 5.10-lts Linux kernel version.[1]

[1]Tests and scripts are available at: netlab.unimore.it/sw/TSQ-NtwL.zip.

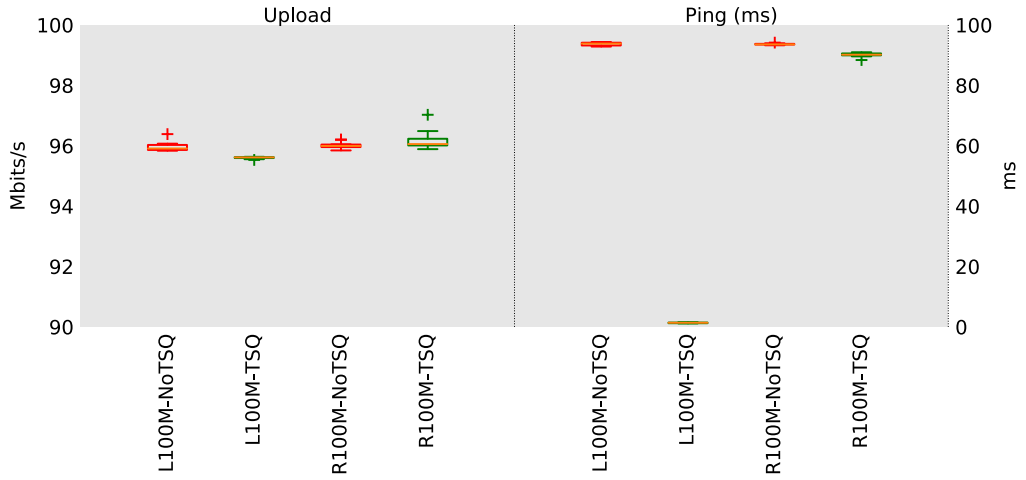
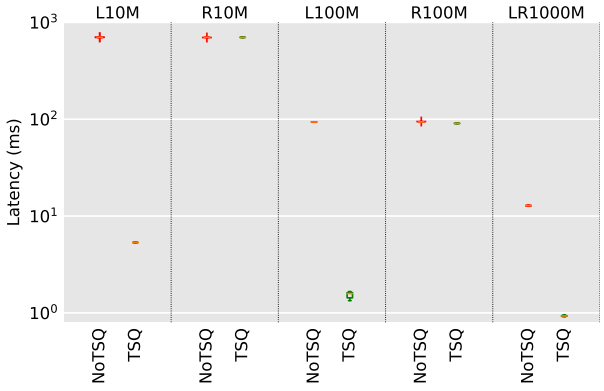
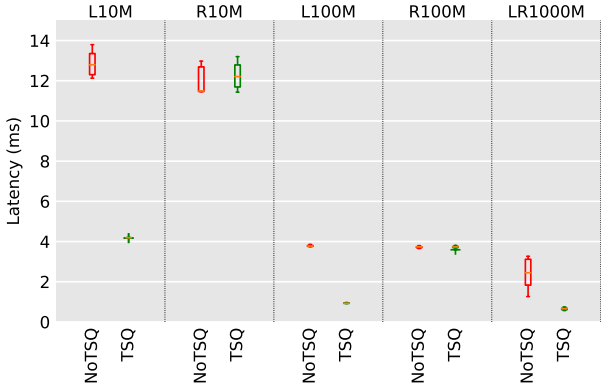


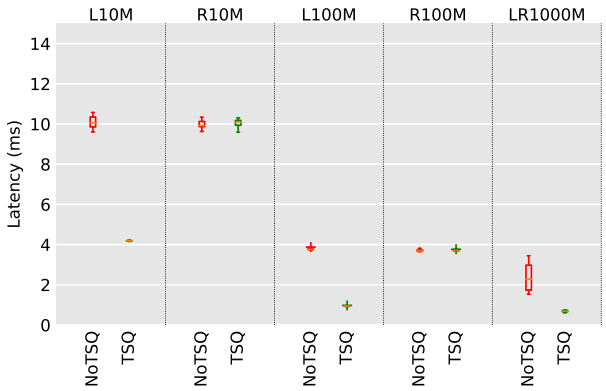
FIGURE 4. Throughput and RTT on different wired bottlenecks: TSQ vs. NoTSQ.



(a) CUBIC and PFIFO.



(b) BBR and PFIFO.

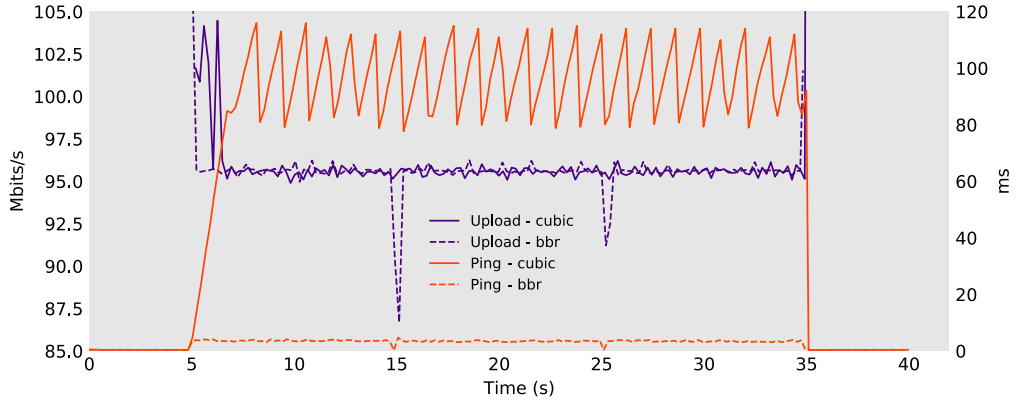


(c) BBR and FQ-CoDel.

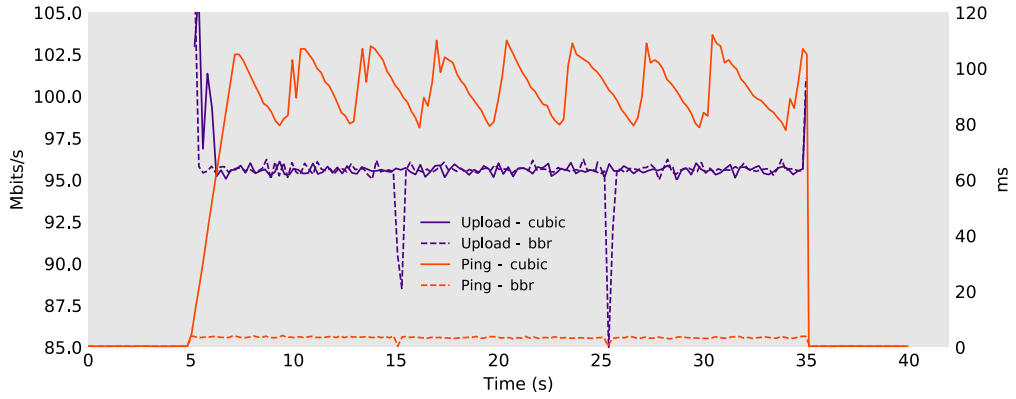
FIGURE 5. TCP RTT: Different wired bottlenecks with TCP BBR or CUBIC.

IV. RESULTS

Our tests are performed using the Flent [14] tool, running four TCP uploads from `C` to `S` for 30 seconds with a concurrent `ping` flow. The core parameter changed between each test is the TSQ size; indeed, we developed a Linux kernel patch[1] to allow us to change the standard TSQ dynamic size of 1 ms of data at the current rate, with a value from 1 to 8 ms of data at the current rate. We also allowed the possibility to disable the TSQ mechanism, naming this strategy NoTSQ in our results. The TCP congestion control algorithms used in our tests are TCP Cubic and TCP BBR to evaluate the TSQ performance in the presence of a loss-based and a delay-based variant, respectively. Moreover, we also evaluated two possible queueing disciplines, FQ-CoDel



(a) R100M, NoTSQ.



(b) R100M, TSQ.

FIGURE 6. Throughput and RTT of TCP BBR and CUBIC on a remote bottleneck.

and PFIFO,[2] which is a priority scheduler with three possible software queues. PFIFO is still the default queueing discipline in several Linux distribution and networking nodes, like the Raspberry PI model 4, waiting for FQ-CoDel to replace it incrementally. We did not focus on the FQ queueing discipline, historically associated with BBR; this is because, before the kernel version 4.17, BBR could not deploy a proper TP mechanism by itself and was recommended to be used in conjunction with FQ. The use of FQ indeed helps in terms of global TCP Pacing, adding an extra CPU usage for this task as shown in [10]. Anyway, in our analysis, we focus on network performance, where throughput and latency need to be controlled through the default packet schedulers or AQM techniques offered by current Linux kernels. As a matter of fact, the usage of FQ is not common anymore since, from the 4.17 kernel, BBR effectively implements its pacing system without needing FQ anymore.

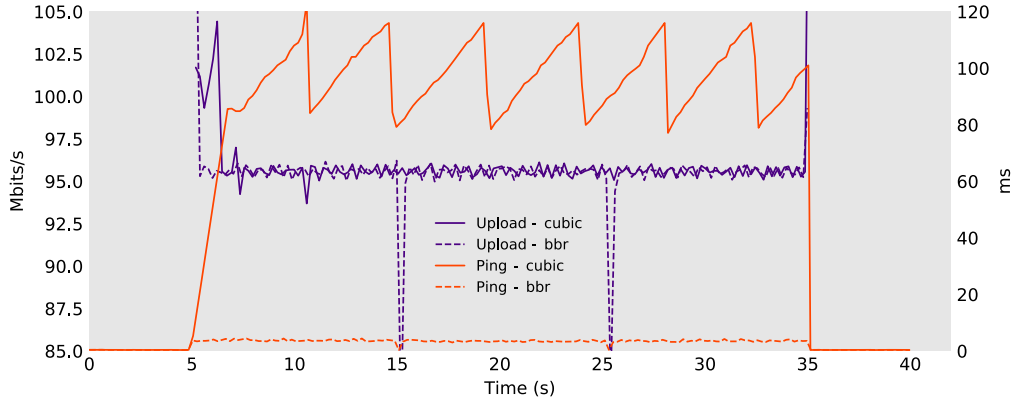
A. WIRED RESULTS

We first introduce the results obtained with the wired version of our testbed of Figure 3. All the wired tests have been performed using the standard TSQ configuration or the NoTSQ one with the mechanism disabled.

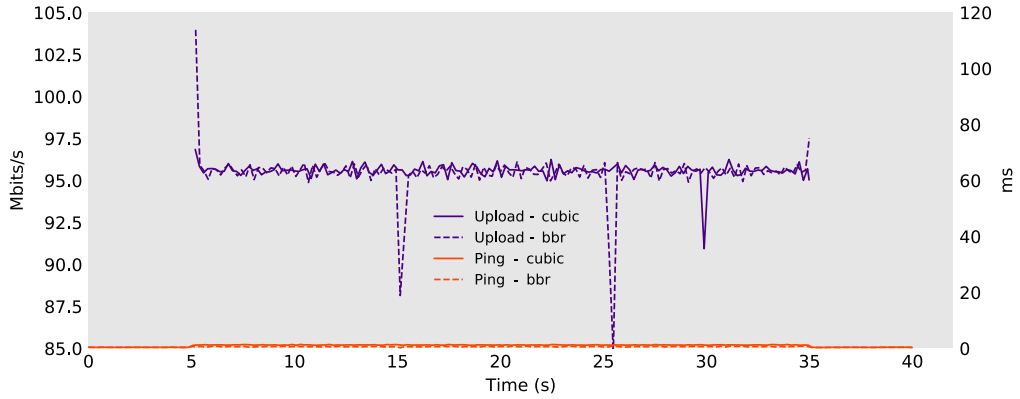
[2]The queueing discipline name in the `tc` package is `pfifo_fast`.

Figure 4 shows the throughput and the latency of a simple TCP Cubic stream from `C` to `S`, with PFIFO used as queueing discipline in the two possible bottlenecks tested: `L100M` and `R100M`. We used PFIFO instead of FQ-CoDel to enhance the sole impact of TSQ on the latency reduction. All our results are in candlestick format; the top and the bottom of the boxes represent the 90th and the 10th percentiles of the data, respectively, while the solid line into the box represents the median data value. One clear evidence of Figure 4 is the remarkable impact of TSQ when the bottleneck is local, because the dominant latency contribution is the queueing delay caused by the sender node queues, which is limited by the TSQ mechanism. It is important to note that this remarkable latency drop is performed maintaining the throughput close to 96 Mbit/s, like in all the other configurations. On the other hand, when the bottleneck is remote, the presence of TSQ only marginally mitigates the end-to-end latency.

Considering that TSQ does not impact a wired bottleneck throughput, we now focus only on the TCP RTT performance of all the five wired bottlenecks tested in Figure 5. Moreover, we also include the impact of a different TCP congestion control and a different queueing discipline, BBR, and FQ-CoDel. Figure 5a is an extension of Figure 4 that embraces also the `L10M`, `R10M` and `LR1000M` bottleneck configurations, focusing on the TCP RTT instead of the



(a) L100M, NoTSQ.



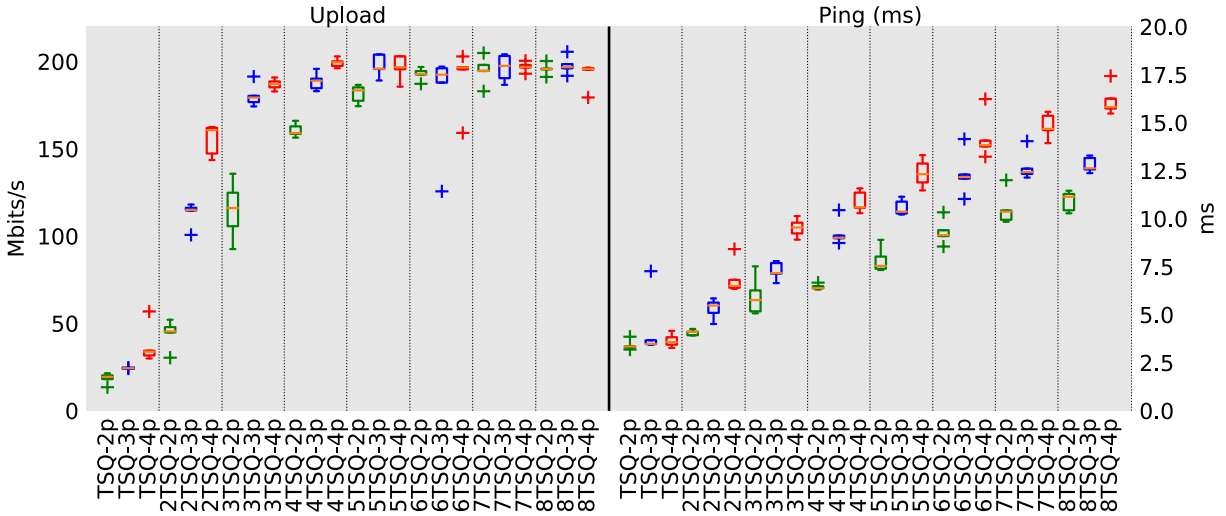
(b) L100M, TSQ.

FIGURE 7. Throughput and RTT of TCP BBR and CUBIC on a local bottleneck.

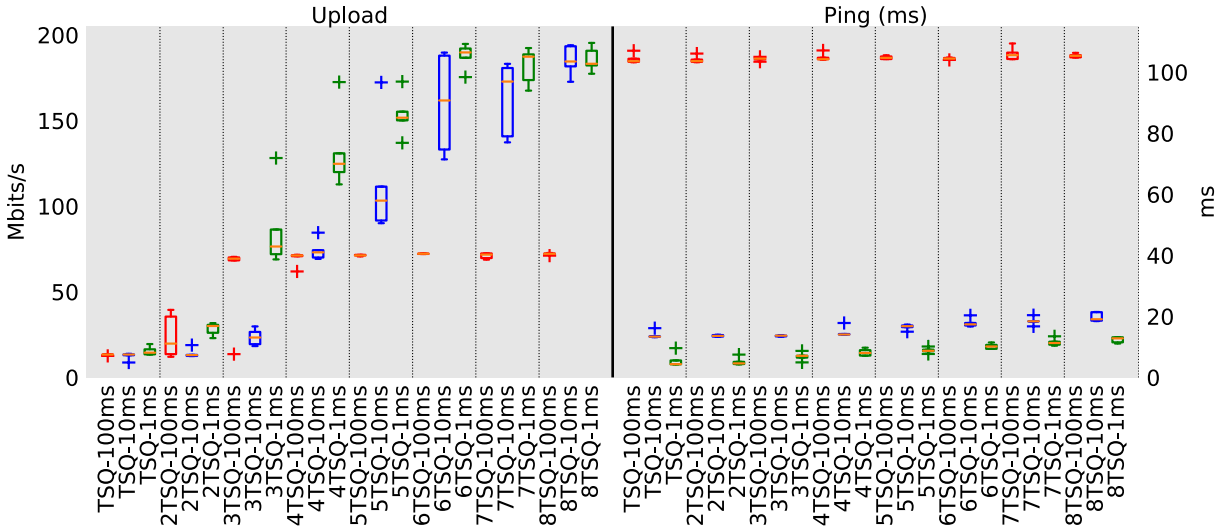
ICMP `ping` RTT. The introduction of TSQ reduces by two orders of magnitude the TCP RTT in `L10M` and `L100M`, where the bottleneck is imposed on the local network through the HTB filter. In the `LR1000M`, instead, where the interfaces are not software-limited, and the HTB filter is disabled everywhere, the TCP RTT reduction introduced by TSQ is of one order of magnitude, which is again a remarkable result.

Moving from Figure 5a to Figure 5b, only the TCP congestion control is changed from CUBIC to BBR. In this case, the impact of TSQ is still observable but with a smaller impact of 10 ms and 3 ms in the `L10M` and `L100M` configurations, respectively. Even in this case, removing the bottlenecks with the `LR1000M` configuration leads to a smaller TSQ impact of a couple of ms. Finally, Figure 5c reports the results of BBR in conjunction with FQ-CoDel, which is very similar to the effects of CUBIC with FQ-CoDel, not reported here. The differences with Figure 5b are the maximum latency in the `R10M` configuration and in the `L10M` with NoTSQ, in which FQ-CoDel imposes a maximum queueing delay at the bottleneck that results in a global TCP RTT of 10 ms.

To conclude the discussion on wired bottlenecks, we also present Figures 6 and 7, which include the same tests of Figure 4, adding also BBR as a congestion control algorithm. Figure 6 reports the throughput and the RTT of CUBIC and BBR on a remote bottleneck operating at 100 Mbit/s with a PFIFO queueing discipline; even though throughput performance is similar, the RTT is remarkably different, and the shape of the curves helps to understand the big difference between the two congestion controls. TCP CUBIC operates by waiting for the loss feedback from the network, filling the bottleneck queue up to the packet drops. Indeed, the shape of the RTT curve presents several peaks corresponding to the maximum queueing delay when the bottleneck queue is full and some minimum peaks corresponding to the new starting congestion window of CUBIC in response to the loss. The behavior of BBR, instead, is different due to the model-based nature of the congestion control. It is possible to notice the draining spikes happening every 10 s, which correspond to the `probe_RTT` phase of BBR reported in Figure 2. Since the bottleneck of Figure 6 is remote, the presence or not of TSQ moving from Figure 6a to Figure 6b does not impact the results. The situation is different in Figure 7, where the bottleneck is local, and packets get accumulated in the sender's NIC. If the TSQ algorithm is not active (Figure 7a), the behavior of the system is logically equivalent to the case of a remote bottleneck since there is no way to control the number of packets at the NIC, and both TCP CUBIC and TCP BBR operate like in Figure 6a. If the TSQ is active, instead,



(a) Different TSQ and TP rates.



(b) Different TSQ settings and base RTTs.

FIGURE 8. **Throughput and RTT on the `ath9k` wireless bottleneck.**

TCP CUBIC completely changes its behavior because TSQ forbids TCP CUBIC to accumulate packets in the bottleneck queue (which is the local NIC's queue). The result is that TCP CUBIC, and any loss-based TCP variant generally, would operate ruled by the TSQ interrupts, maintaining a minimum NIC's queue usage like the BBR congestion control.

B. Wi-Fi RESULTS

The TSQ mechanism has been shown to break the frame aggregation logic of Wi-Fi technologies in [8], so we investigate the impact of different TSQ sizes in conjunction with different TP ratios and different base-RTT flows. We configured 3 different TP ratios by changing the congestion avoidance pace variable `tcp_pacing_ca_ratio`, from the default value of 1.2 to 1.3 and 1.4, naming these configurations `2p`, `3p`, and `4p`, respectively. For what concern, instead, the different RTT flows, we changed the base RTT through the `netem` package at the `AP` with three possible flows configurations, at 1, 10, and 100 ms of base RTT.

Figure 8a shows the results of four TCP CUBIC uploads with our eight different TSQ configurations and the three possible pacing ratios, while Figure 8b shows the results of the same test with a standard pacing ratio and three possible base-RTT. In this set of experiments, we selected only TCP CUBIC since TCP BBR ignores the global TP variable changes. These two plots are presented together to highlight the relation between TP and RTT, which both affect the number of packets that the TCP socket can enqueue and, consequently, the throughput. Theoretically, from the combination of Algorithms 1 and 2, halving the RTT has the same effect of doubling the TP rate.

The values reported in green in Figure 8 are almost the same, due to a slightly different base-RTT of the first experiment, under 1 ms. In Figure 8a, it is possible to see that increasing the pacing rate (blue and red) has the effect of increasing the throughput on the Wi-Fi path and increasing the latency as well. This result is justified by Algorithm 2; indeed, the higher is the TP rate, the larger is the number of packets enqueued in the NIC, which allows for larger aggregates increasing the throughput, despite a latency increment. Once the initial TSQ value is greater than 4TSQ, increasing the TP ratio has the sole effect of increasing the latency since the Wi-Fi bottleneck has already been saturated with the maximum available frame aggregation.

In Figure 8b, on the other hand, it is possible to see that a higher RTT has the same effect as a pacing reduction. This effect can be observed moving from 1 ms of base RTT to 10 ms of base RTT, where the former configuration registers higher throughput with respect to the latter. With a base RTT of 100 ms instead, the delivery rate is too low, and the final throughput result is not optimal, even relaxing the TSQ constraints. This effect is justified because the high RTT forces a low rate, considering Algorithm 1. Consequently, even relaxing the TSQ value in Algorithm 2 does not allow the rate to grow enough to form larger aggregates and discover higher throughputs available on the Wi-Fi channel.

V. CONCLUSION

In this paper, we evaluated that TSQ alone significantly impacts latency when the bottleneck is local. The reason is due to the nature of TSQ, which limits the amount of data that each socket can enqueue, or, in other words, limits the bottleneck queue size if the bottleneck is the sender's NIC. The latency reduction effect is mitigated when algorithms like BBR and FQ-CoDel are deployed, but it is still observable. These results pose the TSQ mechanism in a critical position when dealing with network performance. Network simulators will have to include this mechanism to maintain high fidelity results compared with real systems. Nevertheless, the TSQ has a different impact on a local Wi-Fi bottleneck with respect to a local wired bottleneck; the latency reduction is coupled with a non-optimal throughput if the limit imposed by TSQ is too strict. Simultaneously, TP and the base RTT roles impact the Wi-Fi performance because they modify the TSQ limit, and the higher the TP, or the smaller the base RTT, the higher the throughput will be.

REFERENCES

[1] T. Hoeiland-Joergensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet. (Jan. 2018). *FlowQueue-CoDel*. [Online]. Available: https://tools.ietf.org/html/rfc8290
[2] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *Commun. ACM*, vol. 60, no. 2, pp. 58–66, 2017.
[3] A. Saeed, N. Dukkipati, V. Valancius, V. Lam, C. Contavalli, and A. Vahdat, "Carousel: Scalable traffic shaping at end hosts," in *ACM SIGCOMM*, 2017, pp. 404–417.
[4] B. Stephens, A. Singhvi, A. Akella, and M. Swift, "Titan: Fair packet scheduling for commodity multiqueue NICs," in *Proc. USENIX Annu. Tech. Conf.*, 2017, pp. 431–444.
[5] B. Briscoe, A. Brunstrom, A. Petlund, D. Hayes, D. Ros, I.-J. Tsang, S. Gjessing, G. Fairhurst, C. Griwodz, and M. Welzl, "Reducing internet latency: A survey of techniques and their merits," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 3, pp. 2149–2196, 3rd Quart., 2016.
[6] Y. Zhao, A. Saeed, E. Zegura, and M. Ammar, "zD: A scalable zero-drop network stack at end hosts," in *Proc. CoNEXT*, 2019, pp. 220–232.
[7] Y. Guo, F. Qian, Q. Chen, Z. Morley Mao, and S. Sen, "Understanding on-device bufferbloat for cellular upload," in *ACM SIGCOMM*, vols. 14–16, 2016, pp. 303–317.
[8] C. A. Grazia, N. Patriciello, T. Hoiland-Jorgensen, M. Klapez, M. Casoni, and J. Mangues-Bafalluy, "Adapting TCP small queues for IEEE 802.11 networks," in *Proc. IEEE 29th Annu. Int. Symp. Pers., Indoor Mobile Radio Commun. (PIMRC)*, Sep. 2018, pp. 1–6.
[9] C. A. Grazia, "IEEE 802.11n/AC wireless network efficiency under different TCP congestion controls," in *Proc. Int. Conf. Wireless Mobile Comput., Netw. Commun. (WiMob)*, Oct. 2019, pp. 1–6.
[10] Y. Zhao, A. Saeed, M. Ammar, and E. Zegura, "Scouting the path to a million-client server," in *Passive and Active Measurement*. Cham, Switzerland: Springer, 2021, pp. 337–354.
[11] C. A. Grazia, "Future of TCP on Wi-Fi 6," *IEEE Access*, vol. 9, pp. 107929–107940, 2021.
[12] N. Mareev, D. Kachan, K. Karpov, D. Syzov, and E. Siemens, "Efficiency of BQL congestion control under high bandwidth-delay product network conditions," in *Proc. Int. Conf. Appl. Innov. (IT)*, 2019, vol. 7, no. 1, pp. 19–22.
[13] T. Høiland-Jørgensen, M. Kazior, D. Taht, P. Hurtig, and A. Brunstrom, "Ending the anomaly: Achieving low latency and airtime fairness in WiFi," in *Proc. USENIX ATC*, 2017, pp. 139–151.
[14] T. Hoeiland-Joergensen, C. A. Grazia, P. Hurtig, and A. Brunstrom, "Flent: The flexible network tester," in *Proc. 11th EAI Int. Conf. Perform. Eval. Methodol. Tools (ValueTools)*, 2017, pp. 120–125.

CARLO AUGUSTO GRAZIA (Member, IEEE) received the Ph.D. degree from the Department of Engineering Enzo Ferrari (DIEF), University of Modena and Reggio Emilia (UNIMORE), in 2016. He is currently an Assistant Professor holding the course automotive connectivity with UNIMORE. He has been involved in the EU FP7 Projects E-SPONDER and PPDR-TC. His research interests include computer networking, with an emphasis on wireless networks, queueing algorithms, and V2X.

MARTIN KLAPEZ received the Ph.D. degree from DIEF, UNIMORE, in 2017. He is currently a Postdoctoral Research Fellow with UNIMORE. He has collaborated with the Italian Nanoscience National Research Center S3 and he has been involved in the EU FP7 Project PPDR-TC. His research interests include verge around network softwarization, public safety networks, and safety-related V2X systems.

MAURIZIO CASONI (Senior Member, IEEE) received the M.S. (Hons.) and Ph.D. degrees in electrical engineering from the University of Bologna, Italy, in 1991 and 1995, respectively. In 1995, he was with the Computer Science Department, Washington University in St. Louis, MO, USA, as a Research Fellow. He is currently an Associate Professor of telecommunications with DIEF, UNIMORE, Italy. He has been responsible at UNIMORE for the EU FP7 Projects E-SPONDER and PPDR-TC.